Software Implementation and Testing Document

For

Group <6>

Version 2.0

Authors:
Frankie Messina
Zach Porcoro
Raul Rodriguez
Andrew Stade
Peter Vasiljev

1. Programming Languages (5 points)

- Javascript (with JSX)
  - Required for building React components and handling web behavior. Used in the majority of the source files.
- Additional languages (not exactly programming languages)
  - HTML
    - Required for web development. The elements that make up the content of our website are defined within JSX elements that are very similar to HTML syntax. Although we aren't developing too heavily within any html files, we are still required to utilize many of the same principles needed to write HTML.
  - CSS
    - Required for styling the content of the website. We have external stylesheets that will hold any CSS styling rules so that they can be applied to various elements as needed.

2. Platforms, APIs, Databases, and other technologies used (5 points)

- Platforms
  - Web-based
- Front-end Frameworks
  - React
- APIs
  - Firebase Authentication: Used in the Login and Registration components to create and log into user's accounts. Also used in every page to make sure that the user is actually logged in.
- Databases
  - Firestore: Used in any components where they have to retrieve or save the user's data.
- Libraries
  - MUI (Material-UI) for our React components such as buttons, text fields, etc.
  - React router
- Other technologies
  - VSCode with git source control
  - Node package manager
  - Node.js

3. Execution-based Functional Testing (10 points)

All currently implemented functional requirements have been tested manually as they were implemented. We initially intended to build unit tests for our components to test if the components would meet the specifications, but this idea was pushed to the next increment. The testing that we did perform was more influenced by the idea of black-box testing in the sense that we focused on making sure our components performed the expected functionality.

4. Execution-based Non-Functional Testing (10 points)

We mentioned in version 1.0 of the RD document that one of the main security concerns was the usage of local storage to authenticate users. We implemented a way to verify user authentication on any pages that require it through the usage of a Firebase authentication listener. We tested this security measure by manually adding an uid to local storage before attempting to access a page that requires authentication. Our application was able to prevent the user from accessing those pages because Firebase was unable to verify if the user is authenticated, which causes a redirection to the login screen. The performance of our application was initially subpar after we noticed that the pages were unnecessarily getting refreshed whenever the user clicked on a link. We corrected this issue by using an alternative method of navigation, which improved the performance of our application.

5. Non-Execution-based Testing (10 points)

Whenever we created a pull request, we made sure to have at least one other group member review the changes and approve them. We would also frequently check out other member's branches and view the progress that they had made. Finally, every once and a while we would get in a call together and quickly walk through what features we implemented and how. This allowed us the opportunity to make adjustments to our code if necessary.